一种高效的连续不确定 XML 小枝模式匹配算法*

张晓琳，吕　庆，刘立新，郑春红
（内蒙古科技大学 信息工程学院，包头 014010）

摘　要：针对目前不确定 XML 小枝模式匹配算法均基于归并，易造成很大的空间和时间浪费问题，提出基于 P-文档模型的连续不确定 XML 的非归并的小枝模式匹配算法。算法在节点入队列和出队列时分别进行过滤剪枝操作，减少待处理节点的个数，匹配过程使用相互关联的链表存储中间结果，不需要归并。理论分析与实验结果表明，该算法是一种高效的连续不确定 XML 查询算法。

关键词：连续不确定 XML；小枝模式匹配；过滤剪枝；非归并
中图分类号：TP392　　**文献标志码**：A　　**文章编号**：1001-3695(2013)02-0364-03
doi:10.3969/j.issn.1001-3695.2013.02.011

Efficient algorithm of continuous uncertain XML twig pattern matching

ZHANG Xiao-lin, LV Qing, LIU Li-xin, ZHENG Chun-hong
(*School of Information Engineering, Inner Mongolia University of Science & Technology, Baotou Inner Mongolia* 014010, *China*)

Abstract: At present on account of the uncertain XML twig pattern matching algorithm were based on merging, which including many defects such as waste time and space. This paper proposed a new twig pattern matching algorithm based on existing P-document model which was also for continuous uncertain XML, when the nodes got into the queue and got out of the queue, executed filtering pruning operation, in order to reduce the number of pending nodes, matching process using interrelated list to store the intermediate result, this procedure did not need to merge. The theory research and the results of experiment show that this query algorithm is high efficient.

Key words: continuous uncertain XML; twig pattern matching; filter pruning; no-merging

随着人们对数据的采集和处理技术的提高，在很多领域包括电信、军事、金融、物流、经济等，连续不确定性数据被广泛应用，这些数据的不确定性以概率属性的形式在 XML 文档中表示，形成连续不确定 XML。XML 文档以其自描述性好、可扩展性与灵活性高著称，目前已成为数据表现和交换的主流形式，特别适用于可能产生大量不确定性信息的应用中。在 XML 数据管理中查询处理是最重要的操作，目前已经提出了多种 XML 的查询算法，但对于不确定性 XML 的查询处理研究还很少，已有的研究中对于不确定性 XML 大都是基于非归并的查询。在深入研究 XML 查询算法的基础上，本文提出了适合连续不确定 XML 特性的非归并 Twig 查询算法。

1　相关研究

目前对于不确定 XML 或称为概率 XML 的数据管理技术的研究还处于起步阶段。文献[1]中最早提出了概率关系数据与概率 XML 之间的转换方法。对于普通 XML 文档的查询一般是将 XPath 语句转换为结构连接。目前已经提出了很多结构连接算法，但对于连续不确定 XML 的算法研究还很少。从算法思想上分类可以分为基于归并的和非归并的两大类。

早期的对于 XML 文档的查询处理都是基于归并的，如直接归并算法 MPMGJN[2]，这种算法需要重复扫描文档树的节点列表，逐一匹配二元关系，会产生大量的中间结果，造成时间空间上的浪费。文献[3]提出了 Stack-Tree 算法，改进了直接合并连接的 MPMGJN 算法，利用单栈存储候选节点，加快了查找速度，但仍不能避免大量的中间结果。文献[4]提到的 TwigStack 算法考虑从整体上处理小枝，减少二元关系的连接，得到的匹配结果为单枝，而后再合并，减少了中间结果的产生，但仍需合并操作。文献[5]第一次提到了不确定 XML 的小枝模式匹配算法，在 TwigStack 基础上将概率语义运用到查询中，中间结果过多的问题仍未解决。

2006 年，在文献[6]中提出了完全的整体小枝模式匹配算法 Twig^{2}Stack，不需要任何中间结果的归并，该算法通过建立层次栈存储中间结果，达到整体匹配的效果，但需开辟大量的内存空间。2007 年，文献[7]在 Twig^{2}Stack 基础上进行改进，提出 TwigList 算法，使用链表代替层次栈，减少了内存消耗，提高了查询的效率，但算法实现较复杂。目前对于本文提到的连续不确定 XML，还没有任何的整体小枝非归并匹配算法研究。

2　连续不确定 XML 基本概念

2.1　P-文档树模型

目前对于不确定 XML 文档的研究首先是对其建立恰当的数据模型。在文献[8]中提到了概率树模型并研究了对它的

收稿日期：2012-07-13；**修回日期**：2012-08-26　　**基金项目**：国家自然科学基金资助项目(61163015)；内蒙古自然科学基金重点资助项目(20080404Zd21)

作者简介：张晓琳(1966-)，女，内蒙古包头人，教授，博士，主要研究方向为数据库理论与技术(zhangxl@imust.cn)；吕庆(1986-)，男，河北石家庄人，硕士研究生，主要研究方向为数据库理论；刘立新(1983-)，女，内蒙古通辽人，讲师，主要研究方向为数据库理论、数据挖掘；郑春红(1987-)，山东青岛人，硕士研究生，主要研究方向为数据库理论.

化简算法,对于复杂的概率树模型效率不高。本文采用 P-文档模型[9],这种模型由普通节点和分布式节点两种类型的节点组成,该模型将概率值附加于文档树的边上,各节点的概率依赖于其祖先的概率。每个普通节点都有一个标签和一个唯一的标志符,分布式节点仅用于定义产生随机文档(普通 XML 文档)的概率过程。在 P-文档模型中,最常用的有三种类型节点:a)独立类型节点 ind,独立节点在 P-文档树中出现的概率是独立的,不受其他节点的影响;b)互斥类型节点 mux,互斥节点在 P-文档树中只能出现一个其他节点不出现的节点,或者全都不出现;c)连续类型节点 cont,该类节点表示为 cont(D)形式,其中 D 是实数上的概率分布。与其他分布式节点相比,cont 节点仅出现在叶子上。

2.2　P-文档树编码

XML 编码技术对 XML 树中的每一个节点分配一个唯一的数字码值,通过任意两个节点的码值能够直接判断出两个节点之间的结构关系,提高查询的效率。目前已经提出了多种 XML 编码方案,根据编码原理的不同,可以分为区间编码、前缀编码以及素数编码三大类,而对于连续不确定 XML 的编码技术研究还很少。

2.3　小枝模式匹配

XPath 是最常用的 XML 查询语言,包括孩子轴(/)、后代轴(//)以及谓词,XPath 表达式一般被表示成树的形式,也称为小枝模式。

给出一个 XML 文档树 $D(V_D,E_D)$ 和一个小枝模式查询 $Q(V_Q,E_Q)$,其中 V_D、V_Q 分别代表文档树和查询树的节点集合,E_D、E_Q 表示边的集合。在文档树 D 上匹配 Q 的过程就是 Q 的节点到 D 的元素节点映射的过程,表示为 $h:\{x:x\in V_Q\}\rightarrow\{y:y\in V_D\}$,满足下列条件:a)对每一个查询节点 $x\in V_Q$,V_D 中的元素节点 $h(x)$ 与 x 具有相同的标签名,且满足 x 的谓词约束;b)V_D 中的元素节点 $h(u)$ 和 $h(v)$ 之间的关系必须满足 V_Q 中相应查询节点 u 和 v 之间的结构约束,如 *P-C* 关系、*A-D* 关系。

3　连续不确定 XML 小枝模式匹配算法 CUTwigList

该算法的主要思想是:考虑到对于不确定 XML 的查询大多数都是在一定范围内的阈值查询,于是在深度优先遍历文档树节点之后,在匹配过程之前对节点先进行过滤,达到对文档树剪枝的目的。然后依照 Twig 查询树节点,根据提出的改进的素数编码判断节点间的结构关系,建立相互关联的队列,在节点入队列和输出结果之前再根据阈值过滤,遍历序列结束后枚举出结果,整个查询处理过程没有任何中间结果的归并,提高了查询的效率。

3.1　改进的素数编码

定理 1　算术基本定理。任意一个大于 1 的自然数 N 都可以唯一分解成有限个素数的乘积。

这里的分解称为 N 的标准分解式。算术基本定理由两部分内容组成:a)分解的存在性,每一个大于 1 的自然数 N 都能分解成素数因子的乘积;b)分解的唯一性,如果不考虑素数因子排列的顺序,正整数分解为因子乘积的方式是唯一的。

本文根据定理 1,采用改进的素数编码方案。在 P-文档基础上,每一个节点的编码由一个四元组〈prime,primeMulti,pro,path〉构成。其中,prime 代表的是为节点分配的素数码值,编码过程中按层遍历文档树的每一个节点;如果节点 u 是 v 的父亲节点,则 v 的 primeMulti 是节点 u 的 prime 与 primeMulti 的乘积,可以利用 primeMulti 来判定结构关系;pro 代表节点概率特性,标志其存在的概率值,若节点是普通类型的节点,则 pro 的值为 1,若节点是不确定性节点,则 pro 为一个概率数值 $P(0<P<1)$;path 代表从根节点到自身节点路径上的分布节点的类型,P-文档中常用的分布节点有独立类型(ind)的节点、互斥类型(mux)的节点和连续型(cont)节点,cont 节点只出现在叶子节点上(如图 1 中的节点 $N(15,3)$),表示路径时用 i 代表 ind 节点,m 代表 mux 节点。如果路径上不存在分布类型的节点则 path = o,o 代表 ordinary 即普通类型的节点。

1)兄弟关系的判定　若节点 A、B 具有兄弟关系,则 A.primeMulti = B.primeMulti。例如图 1 中(3.9)和(5.9)、(3.27)和(5.27)等。

2)父子关系判定　若节点 A 是 B 的父亲,则 B.primeMulti = A.prime × A.primeMulti。例如图 1 中的(3.3)和(3.9)、(3.9)和(5.27)作为 A、B 节点,其中 9 = 3 × 3,27 = 3 × 9。

3)祖先子孙关系判定　若判断节点 A、B 是否存在祖先后代关系,则利用定理 1 把 B 的 primeMulti 分解成素数因子的乘积,第一个素数因子从小于等于 prime 的数开始,再将这些素数因子转换为字符串 i;然后把 A 的 prime 和 primeMulti 的素数因子转换为字符串 j;最后判断 i 是否包含且以 j 结束。例如,图 1 中 A:(5,9)、B:(7,315)。对于 B 来说,primeMulti 为 315,315 = 7 × 5 × 3 × 3,对应字符串 i = "7533";对于 A 来说对应字符串 j = "533"。显然满足以上的结构关系条件,证明 A 是 B 的祖先节点。

3.2　CUTwigList 算法

算法 CUTwigList(Q)

输入:查询树 Q,其有 n 个节点$\{v_1,v_2,\cdots,v_n\}$。

输出:满足 Q 的所有元素节点。

a) V_i = Travelsal(T);//深度优先遍历树,得到其节点序列

b) Lv_i = CUTwigList-Construct(Q,V_i);

//根据查询树节点,构造所需的链表

c) R = CUTwigList-Enumerate(Q,L);//枚举出匹配结果

d) return R//返回查询结果

CUTwigList(Q,T)是主算法,a)是对文档数进行遍历,得到其节点集合;b)是根据查询树的节点构造链表,存储中间结果;c)d)是匹配的过程,返回匹配的结果。

以上主算法中 CUTwigList-Construct(Q,V_i)是核心的操作。以下为其具体过程:

输入:查询树 Q,其有 n 个节点$\{v_1,v_2,\cdots,v_n\}$。

输出:满足 Q 的所有元素节点。

a) n_i = getNext(V_i);//通过阈值过滤,得到符合条件的节点

b) Stack S = new Stack();//初始化路径栈

c) for(;1 <= i <= n;i++)

List Lv_i = new List(Q);

//根据查询树初始化链表

d) while(n_i != null){

push(S,n_i);//节点入路径栈

toList(S,Q,n_i);//符合查询树的节点入链表

}

Procedure toList(S,Q,n_i)

{

while(S != null)

```
}
for( ;1 < =i < =n;i + + )
{ Vj = Pop(S) ;}
Vk = Lvi . topElement( ) ;
if( ( Vk . primeMulti) . endWith( Vj . primeMulti) )
/ * 根据编码判断节点之间是否满足查询模式结构关系,满足则入队列 * /
Lvi . append( Vj ) ;
Vj . pointer = Vk . primeMulti;
}
```

3.3 CUTwigList 算法示例

下面用图 2 的文档树和图 3 的查询树来说明 CUTwigList 和传统 TwigList 的执行过程。

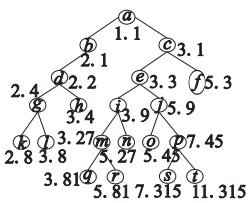

图1 CUPE简易编码树

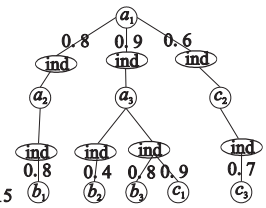

图2 文档树

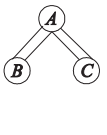

图3 查询树

图 2 的文档树是一棵连续不确定 XML 的 P-文档,包含独立节点 ind、互斥节点 mux 以及叶子节点上的连续节点。图 3 是用户的查询树。

1)CUTwigList 执行

首先执行 Travelsal(T)得到文档树节点的深度遍历序列及其概率值{a_1:1,a_2:0.8,b_1:0.64,a_3:0.9,b_2:0.36,b_3:0.72,c_1:0.81,c_2:0.6,c_3:0.42}。这里假定查询的阈值为 0.5,执行 CUTwigList-Construct(Q,V_i)方法时,通过 getNext( )过滤得到序列{a_1:1,a_2:0.8,b_1:0.64,a_3:0.9,b_3:0.72,c_1:0.81,c_2:0.6},查询算法只在这些节点上进行。设定一个指针指向序列首元素 a_1,a_1 首先入路径栈,然后指针依次后移。节点入栈之前必须保证栈顶元素是其祖先节点,若不是则栈顶元素出栈,继续判断与新的栈顶元素的关系是否满足祖先子孙关系。对于出栈的元素符合查询要求的进入对应的链表,不符合直接丢弃。当查询树的根节点进入链表时,设定指针指向其子孙节点。当路径栈为空时,执行 CUTwigList-Ennumerate(Q,L)开始遍历根据查询树根节点建立的链表,取出匹配结果时考虑互斥类型的节点,将结果中的节点存在互斥关系的节点删除。以图 2 的文档树和图 3 的查询树为例,算法的执行过程如图 4 所示。图 4(a)中,c_1 插入路径栈后栈和链表的状态,b_1、b_3 入链表;(b)中 c_2 插入路径栈后,栈和链表的状态,a_3 入链表设定指针指向子孙节点;(c)中路径栈为空时,栈和链表的状态。最后遍历链表 A 取出匹配结果:$a_3b_3c_1$、$a_1b_1c_1$、$a_1b_1c_2$、$a_1b_3c_1$、$a_1b_3c_2$,因为 b_3、c_1 为互斥节点所以过滤掉 $a_3b_3c_1$、$a_1b_3c_1$;又因为阈值为 0.5 过滤掉 $a_1b_1c_2$(0.384)、$a_1b_3c_2$(0.432),得到最终结果 $a_1b_1c_1$(0.518)。

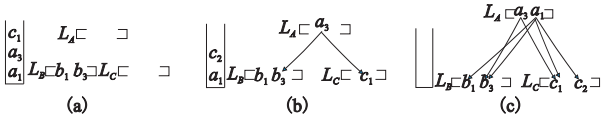

图4 算法执行过程

2)TwigList 执行

得到文档树节点的深度遍历序列及其概率值{a_1:1,a_2:0.8,b_1:0.64,a_3:0.9,b_2:0.36,b_3:0.72,c_1:0.81,c_2:0.6,c_3:0.42}后,直接执行 TwigList 得到所有的匹配结果:$a_3b_2c_1$—0.324,$a_3b_3c_1$—0.576,$a_1b_1c_1$—0.5184,$a_1b_1c_2$—0.384,$a_1b_1c_3$—0.2688,$a_1b_2c_1$—0.2916,$a_1b_2c_2$—0.216,$a_1b_2c_3$—0.1512,$a_1b_3c_1$—0.5832,$a_1b_3c_2$—0.432,$a_1b_3c_3$—0.3024。然后把所有的匹配结果的概率值求出,在根据阈值进行过滤,很明显会产生很多无用的中间结果,造成很大的时间和空间上的开销。

3.4 算法复杂度分析

从构造链表 CUTwigList-Construct(Q,V_i)和枚举输出 CUTwigList-Ennumerate(Q,L)两个阶段来分析算法的时空复杂度。假定通过阈值过滤后 P-文档中剩余的节点数为 n,小枝模式中共有 N 个查询节点,且查询节点的最大度为 D。

1)时间复杂度 阈值过滤后剩余的 n 个节点,在 CUTwigList-Construct(Q,V_i)阶段,每一个节点都要经历一次入/出栈操作,最坏的情况下每一个节点最多需要经历 D 次编码比较,决定其孩子节点是否入栈,此阶段的时间复杂度是 $O(nD)$。在 CUTwigList-Ennumerate(Q,L)阶段,假定最终结果解个数是 R,小枝模式 N 个节点每一个节点都要输出,故此阶段的时间复杂度是 $O(NR)$。所以,CUTwigList 算法的时间复杂度是 $O(nD+NR)$。

2)空间复杂度 在 CUTwigList-Construct(Q,V_i)阶段,最坏情况下的时间复杂度是 $O(nD)$,阈值过滤后剩余的 n 个节点最后都存入链表中。在 CUTwigList-Ennumerate(Q,L)阶段枚举查询结果,空间复杂度仍然是 $O(nD)$。所以,ProTwigList 算法的空间复杂度是 $O(nD)$。

4 实验结果分析

采用 Java 实现提出的 CUTwigList 算法,并与原有的 TwigList 进行对比。实验的硬件环境为:CPU Intel® Core i3 (2.26 GHz),RAM 为 4 GB,操作系统为 64 位的 Windows 7 旗舰版 SP1,实验工具为 Eclipse SDK 3.7.1,JDK 6.0。实验测试所用数据集是在 XML 经典数据集 DBLP 文档结构的基础之上,利用一个随机化的算法随机加入一些不确定性的节点,合成具有不确定性节点的数据集。

实验部分共有三组测试(图 5~7)来对比两种算法。第一组测试条件是相同的文档、相同的查询语句、不同的阈值;第二组测试条件是相同的文档、不同的查询语句、相同的阈值;第三组测试条件是不同的文档、相同的查询语句、相同的阈值。每组实验均重复十次,得到的实验数据采用去掉最大值和最小值,取平均值的方法记录整理。查询用到的查询用例如表 1 所示。

表 1 实验用到的查询用例

name	Twig
Q_1	dblp//article[//title]//year
Q_2	dblp//article[author]//title
Q_3	dblp//inproceedings[title]//year
Q_4	dblp//article[author][//title]//url
Q_5	dblp//inproceedings//author

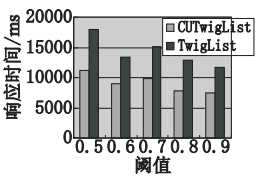

图5 阈值不同时间对比

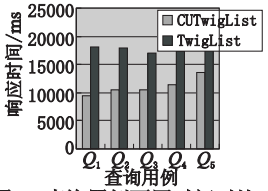

图6 查询用例不同时间对比

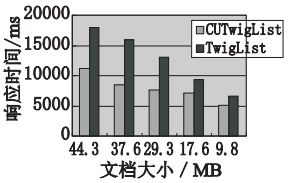

图7 文档大小不同时间对比

第一组测试选取文档大小为 44.3 MB,执行查询语句 Q_1,分别选取了五个不同的阈值测得查询的响应时 (下转第 370 页)

$$\text{s.t. } g_1(\boldsymbol{x}) = 1 - \frac{x_2^3 x_3}{71785 x_1^4} \leqslant 0$$

$$g_2(\boldsymbol{x}) = \frac{4x_2^2 - x_1 x_2}{12566(x_2 x_1^3 - x_1^4)} + \frac{1}{5108 x_1^2} - 1 \leqslant 0$$

$$g_3(\boldsymbol{x}) = 1 - \frac{140.45 x_1}{x_2^2 x_3} \leqslant 0$$

$$g_4(\boldsymbol{x}) = \frac{x_1 + x_2}{1.5} - 1 \leqslant 0$$

其中:$0.05 \leqslant x_1 \leqslant 2, 0.25 \leqslant x_2 \leqslant 1.3, 2 \leqslant x_3 \leqslant 15$。

利用 ICOGA 对拉压弹簧优化设计问题进行求解,并与文献[5]中的方法和文献[10]的 PSO-ABC 算法进行比较,结果如表 3 所示。从表 3 中的结果可知,ICOGA 优于文献[5]中的方法,比 PSO-ABC 算法要稍劣。

表 3 几种算法对拉压弹簧优化设计问题的结果比较

算法	$x_1(d)$	$x_2(D)$	$x_3(P)$	$f(x)$
Belegundu	0.050 000	0.315 900	14.250 000	0.012 833 4
Arora	0.053 396	0.399 180	9.185 400	0.012 730 3
Coello	0.051 480	0.351 661	11.632 201	0.012 704 8
CPSO	0.051 728	0.357 644	11.244 543	0.012 674 7
PSO-ABC	0.050 000	0.374 430	3.200 110	4.867 7e-3
ICOGA	0.050 935	0.351 541	5.009 216	0.012 551 4

4 结束语

本文提出一种改进的约束优化遗传算法。为了维持种群个体的多样性,利用佳点集方法产生初始种群。将种群分为可行子种群和不可行子种群,分别从两个子种群中随机选择个体进行算术交叉操作,使算法从不同搜索方向逼近全局最优解。对可行个体与不可行个体分别采用高斯变异与柯西变异,从而达到了约束优化进化算法的两个目标。两个标准测试问题和两个工程设计优化问题的实验结果表明,该算法具有较强的通用性和稳定性。

参考文献:

[1] RUNARSSON T P, YAO Xin. Stochastic ranking for constrained evolu-tionary optimization[J]. IEEE Trans on Evolutionary Computation,2000,4(3):284-294.

[2] 王翔, 董晓马, 阎瑞霞, 等. 改进 DE/EDA 算法在求解难约束优化问题中的应用研究[J]. 计算机应用研究,2010,27(11):4114-4117.

[3] MICHALEWICZ Z, SCHOENAUER M. Evolutionary algorithm for constrained parameter optimization problems [J]. Evolutionary Computation,1996,4(1): 1-32.

[4] 梁昔明, 龙文, 秦浩宇, 等. 基于种群个体可行性的约束优化进化算法[J]. 控制与决策,2010,25(8):1129-1132.

[5] 龙文. 求解两类优化问题的混合进化算法及其应用[D]. 长沙:中南大学,2011.

[6] 张铃, 张钹. 佳点集遗传算法[J]. 计算机学报,2001,24(9):917-924.

[7] CAI Zi-xing, WANG Yong. A multiobjective optimization based on evolutionary algorithm for constrained optimization[J]. IEEE Trans on Evolutionary Computation, 2006,10(3): 658-675.

[8] MEZURA-MONTES E, COELLO C A C. A simple multimembered evolution strategy to solve constrained optimization problems [J]. IEEE Trans on Evolutionary Computation,2005,9(1): 1-17.

[9] 梁昔明,秦浩宇,龙文. 一种求解约束优化问题的遗传算法[J]. 计算机工程,2010,36(14): 147-149.

[10] 王珂珂,吕强,赵汗青,等. 求解工程约束优化问题的 PSO-ABC 混合算法[J]. 计算机应用研究,2012,29(4):1230-1233.

(上接第 366 页)间。第二组测试同样选取文档大小为 44.3 MB,查询的阈值设定为 0.5,分别执行表 1 所示的五个查询用例。第三组测试查询的阈值设定为 0.5,查询用例为 Q_1,然后选取大小不同的五个文档进行测试。从图 5~7 明显看出在三种情况下 CUTwigList 性能都要优于 TwigList。其原因是 CUTwigList 根据阈值设定了过滤策略,匹配之前减少了所处理节点的个数,大大减少了出入栈的操作;而 TwigList 算法需要遍历整个文档树的节点,匹配过程产生大量无关的中间结果。

5 结束语

本文提出了一种非归并不确定 XML 小枝模式查询算法,即 CUTwigList 算法,扩展了 P-文档使之适用于包含连续节点,采用特定的标志规则对分布节点进行区别,使用小素数对文档树进行编码;查询匹配过程中利用阈值对不符合要求的节点进行过滤,构建结果匹配链表,枚举输出时根据节点互斥属性和阈值再过滤,得到最终的匹配结果。在理论分析和实验测试方面均表明 CUTwigList 具有高效性。进一步的工作是对多维连续分布的编码和查询技术作更深入的研究。

参考文献:

[1] 王建卫,郝忠孝. 概率关系模式与概率 XML 模式转换算法研究[J]. 计算机应用研究,2011,28(2):609-612.

[2] ZHANG C, NAUGHTON J, DeWITT D, *et al*. On supporting containment queries in relational database management systems[C]// Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM Press,2001:425-426.

[3] AI-KHALIFA S, JAGADISH H V, KOUDAS N, *et al*. Structural joins: a primitive for efficient XML query pattern matching[C]//Proc of the 18th International Conference on Data Engineering. Washington DC: IEEE Computer Society,2002:141-152.

[4] NICOLAS B, NICK K, DIVESH S. Holistic twig joins: optimal XML pattern matching[C]//Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2002: 310-321.

[5] LI Ya-wen, WANG Guo-ren, XIN Jun-chang, *et al*. Holistically twig matching in probabilistic XML[C]//Proc of the 25th International Conference on Data Engineering. Washington DC: IEEE Computer Society,2009:1649-1656.

[6] CHEN Song-ting, LI Hua-gang, TATEMURA J C, *et al*. Twig^{2}Stack: bottom-up processing of generalized-tree-pattern queries over XML documents[C]//Proc of the 32nd International Conference on Very Large Data Bases. 2006:283-294.

[7] QIN Lu, YU J X, DING Bo-lin. TwigList: make twig pattern matching fast[C]//Proc of the 32nd International Conference on Very Large Data Bases. 2006:313-324.

[8] 王建卫,郝忠孝. 一种概率 XML 树化简算法[J]. 计算机应用研究,2010,27(12):4541-4547.

[9] ABITEBOUL S, CHAN T H, KHARLAMOV E. Aggregate queries for discrete and continuous probabilistic XML[C]//Proc of the 13th International Conference on Database Theory. New York: ACM Press,2010:50-61.